Chapitre 1

GÉNÉRALITÉS

1- INTRODUCTION

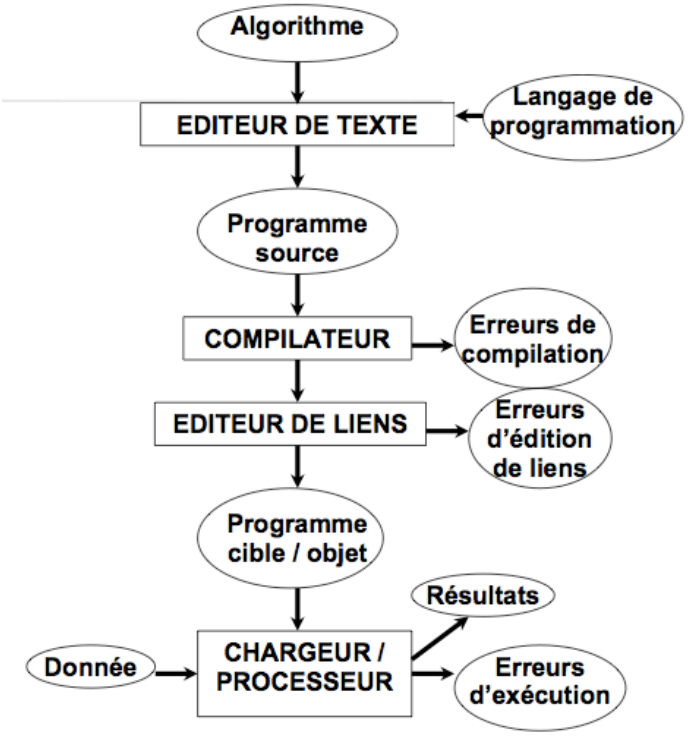
- Tout programme utilise un outil essentiel à la réalisation des programmes informatiques : LE COMPILATEUR

- Un **compilateur** est un logiciel qui traduit un programme écrit dans un langage de haut niveau (par le programmeur) en instructions exécutables (par un ordinateur).

Objectif du cours :

- Présenter les principes de base à la réalisation de compilateurs (analyse lexicale, syntaxique et sémantique et génération de code)

- Utiliser des outils générateurs d'analyseurs lexicaux et syntaxiques pour la création de compilateurs.



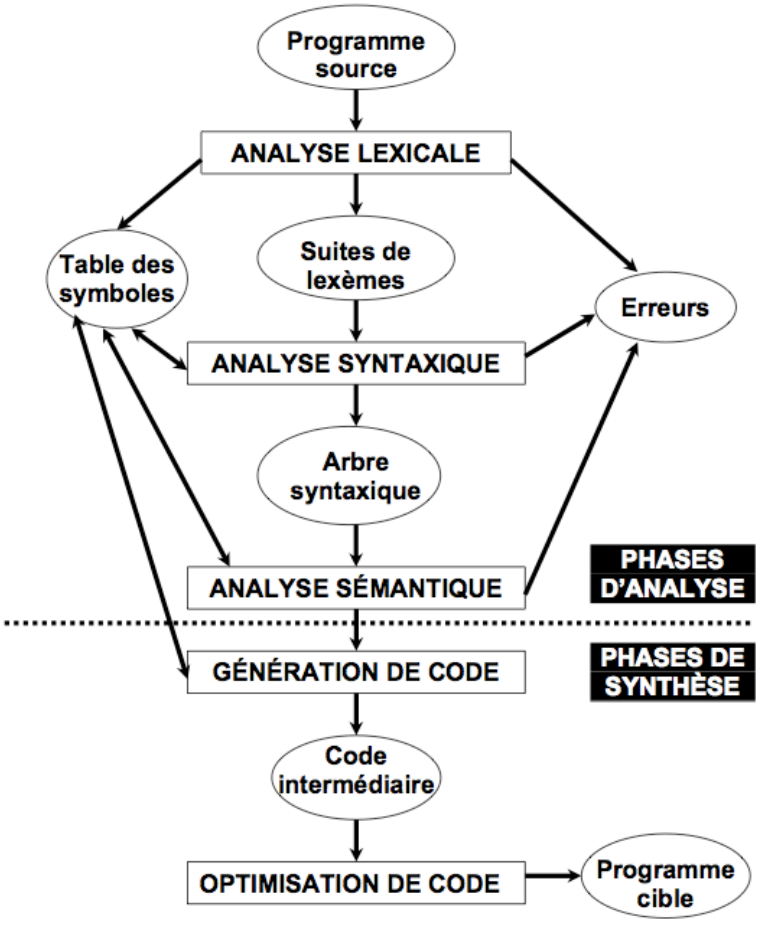
2- RÔLES DE LA COMIPLATION

- Un compilateur est un programme qui traduit un programme écrit dans un premier langage (langage source) en un programme équivalent écrit dans un autre langage (langage cible).

Le rôle du compilateur peut être aussi :

- Traduction d'un LHN vers un autre LHN (PASCAL → C, JAVA → C++,…),
- Traduction d'un LBN vers un LHN (piratage, récupération de vieux logiciels,…),
- Traduction d'un langage quelconque vers un autre langage quelconque pas forcément un langage de programmation (WORD → HTML, PDF → PS,…)

3- STRUCTURE D'UN COMPILATEUR



4 - PHASES DE COMPILATION

- **PHASE D'ANALYSE (PARTIE FRONTALE / EN AVANT)**
  Reconnaître les variables, les instructions, les opérateurs et élaborer la structure syntaxique (arbre) du programme source ainsi que certaines propriétés sémantiques.

- **PHASE DE SYNTHÈSE ET DE PRODUCTION**
  Construire le programme cible à partir de la représentation Intermédiaire (arbre) issue de la phase d'analyse.

4.1 PHASES D'ANALYSE

4.1.1 Analyse Lexicale (Linéaire)

Principe :

- Reconnaître les « types » des « mots» lus.
  - Les caractères sont regroupés en **unités lexicales**.

Exemple : Énoncé: ***for i :=1 to vmax do a :=a+i;***

Suite de lexèmes et d'unités lexicales :

for	***: MOT_CLE***
i	***: IDENT***
:=	***: AFFECT***
1	***: ENTIER***
to	***: MOT_CLE***
vmax	***: IDENT***
do	***: MOT_CLE***
a	***: IDENT***
:=	***: AFFECT***
a	***: IDENT***
+	***:OP_ARITH***
i	***: IDENT***
;	***: SEP***

4.1.2 Analyse Syntaxique (Hiérarchique / Grammaticale)

Principe :

- Vérifier que les unités lexicales sont dans le bon ordre défini par le langage,
  - Regrouper les unités lexicales en structures grammaticales (déclarations, expressions, instructions, appels de fonctions…),

- Découvrir la structure du programme
  - Produire une représentation sous forme d'arbre de la suite des unités lexicales obtenues lors de la phase précédente.

Exemple : En C, une sélection simple doit se présenter sous la forme :
If (expression) instruction

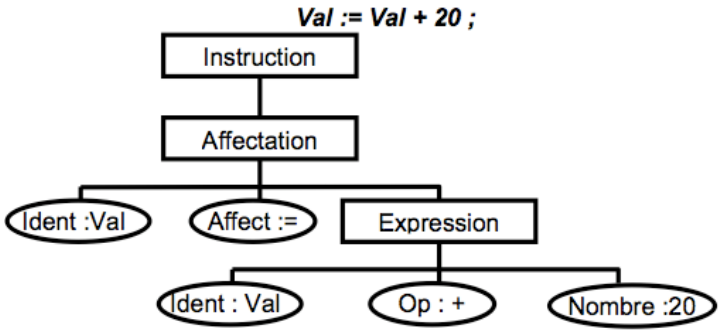
Si l'analyseur syntaxique reçoit la suite d'unités lexicales :

MOT_CLE IDENT OP_REL ENTIER…

Il doit signaler que ce n'est pas correct car il n'y 'a pas **(** juste après le **if.**

L'analyseur syntaxique produit un arbre syntaxique.

Exemple : L'arbre suivant représente la structure de la phrase :



4.1.3 Analyse Sémantique (Contextuelle / Vérification de type)

Principe :

- Vérifier que les variables ont un type correct.
  - Cette opération s'effectue en parcourant l'arbre syntaxique et en vérifiant à chaque niveau que les opérations sont correctes.

Exemple : ***for i :=1 to vmax do a :=a+i;***

Il faut vérifier que la variable 'i' possède bien le type 'entier', et que la variable 'a' est bien un nombre.

4.2. PHASES DE SYNTHÈSE (PRODUCTION)

4.2.1 Génération de Code

Principe :

- Produire les instructions en langage cible (issu d'une machine virtuelle / abstraite).
  - Une **machine virtuelle** ressemble à une machine réelle, mais cette machine n'existe pas en tant que telle et doit être simulée par une machine réelle.

    L'avantage : portabilité et élimination de certains détails d'implémentation inutiles.

1- Production des instructions pour une machine virtuelle.
2- Traduction en des instructions directement exécutables par la machine réelle sur laquelle on veut que le programme s'exécute.

Exemple : Code généré pour une (pseudo) machine pour l'énoncé :

for i :=1 to vmax do a :=a+i;

```
        var_a           A0000     ; Étiquettes des variables
        var_i           A0001
        var_vmax        A0002

        ...                       ; Code du programme mov
        var_i, 1
loop:
        mov A0, (var_i)           ; comparaison i >= vmax jge A0,
        (var_vmax), finFor        ; si vrai aller en finFor mov A0,
        (var_a)                   ; calcul de a+i
        add A0, A0, (var_i)
        mov var_a,A0              ; a := a+i
        mov A0, (var_i)           ; on incrémente i càd i:=i+1 add A0,
        A0, 1
        mov var_i, A0
        jmp loop                  ; et on continue la boucle finFor:
...
```

4.2.2 Optimisation de Code

Principe :

- Améliorer le code produit afin que le programme résultant soit plus rapide.
    - Améliorations qui ne dépendent pas de la machine cible (*élimination des calculs inutiles faits en double,…*),
    - Améliorations qui dépendent de la machine cible (*remplacement des instructions générales par des instructions plus efficaces et plus adaptées, utilisation optimale des registres, …*).

4.3. PHASES PARALLÈLES

4.3.1 Gestion de la table des symboles

- **La table des symboles** est utilisée pour stoker les informations concernant les identificateurs du programme source (*Ex. : leur type, leur emplacement mémoire, nombre, type et mode de passage des paramètres d'une fonction,…*).

- Le remplissage de cette table (collecte d'informations) a lieu lors des phases d'analyse. Son contenu est nécessaire lors des analyses syntaxique et sémantique ainsi que lors de la génération de code.

4.3.2 Gestion des erreurs

- Chaque phase peut rencontrer des erreurs. Il faut détecter et informer l'utilisateur le plus précisément possible (type d'erreur, cause, position dans le programme source,…).

- Plusieurs types d'erreurs :
    - Erreurs lexicales : fautes d'orthographe dans un identificateur ou dans un mot clef, caractères illégaux,…
    - Erreurs syntaxiques : manque de parenthèses dans une expression arithmétique, erreurs de structures de blocs, manque de séparateurs,…
    - Erreurs sémantiques : identificateur non déclaré, incompatibilité entre opérateurs et opérandes,…
    - Erreurs logiques : erreurs arithmétiques : division par zéro, racine carré d'un nombre négatif, dépassement des limites d'un tableau, boucle infinie,…

- Après avoir détecté une erreur, il faut la traiter de telle manière que le compilateur puisse continuer et que d'autres erreurs puissent être détectées.

5. EXEMPLE DE TRADUCTION D'UNE INSTRUCTION